A Java IDP for a Knowledge Base System

Nick Calus Joost Vennekens

Lessius Mechelen – Campus De Nayer
J. De Nayerlaan 5, 2860 Sint-Katelijne-Waver

1 Practical information

- This software was developed by Nick Calus for his Master's thesis
- System requirements: a recent Java Runtime Environment on a platform that supports the IDP system (Windows, Linux, Mac)
- Duration of demo: 15 minutes

2 Summary

Many software applications today are designed according to the Model-View-Controller paradigm, using a variety of supporting tools. Automic GUI builders automate much of the work needed to build a View component, and application architectures such as JEE or .NET provide also support for making Controllers. Typically, however, the Model component still consists of a number of hand-written classes. An interesting alternative is offered by the *knowledge base* paradigm, in which the programmer directly writes down his knowledge about the problem domain, instead of "compiling" it into imperative program code. General reasoning algorithms can then be applied to this knowledge in order to actually implement the desired behaviour of the system.

To make this work in practice, though, some challenges still remain. First, there is a need for expressive knowledge representation languages that make it easy for the programmer to write down his domain knowledge. Second, there must be efficient implementations of reasoning algorithms for this language. And finally, there also needs to be support for linking the knowledge base and the output of the reasoning algorithms to a GUI with as little effort as possible.

The software showcased in this demo provides an API [2] to link user interfaces written in Java to the IDP knowledge base system [1]. The language used by this system is an expressive extension of classical first-order logic, that includes a type system, aggregates and inductive definitions. For this language, the system implements the inference task of model expansion, which is a natural fit with the type of inference needed in, e.g., configuration software [3]. Finally, the API allows this powerful knowledge base system to be attached to a Java GUI with minimal effort. In particular, the API has a "simple" mode of operation, where an entire user interface is built automatically from the knowledge base. There is of course also the possibility to first create a user interface, and then quickly link this to the knowledge base. Finally, there is also an "advanced" mode which provides the programmer with a lot of control over the operation of the API, allowing him to customize most of its behaviour and appearance.

To illustrate, Figure 1 shows a screenshot of a configuration system that allows customers to create their own bicycle. The GUI for this system was made with the standard GUI builder included in the NetBeans IDE. The software does not just allow the user to make his wishes known, but actively tries to help him make a valid set of selections. For instance, in the above screenshot, the "Fietstype" (type of bicycle) has been automatically selected by the system, because Grandma Bikes are the only ladies' bicycles available for size 190cm-200cm. In other words, this is the only bicycle type still compatible with the choices the user has already made. For the "Frame", something similar is going on, since the system has figured out that "Centurion Boulevard" is the only valid option left.

None of this behaviour has been hand-coded, though. It is all implemented by means of an IDP knowledge base. This knowledge base contains formulas such as:

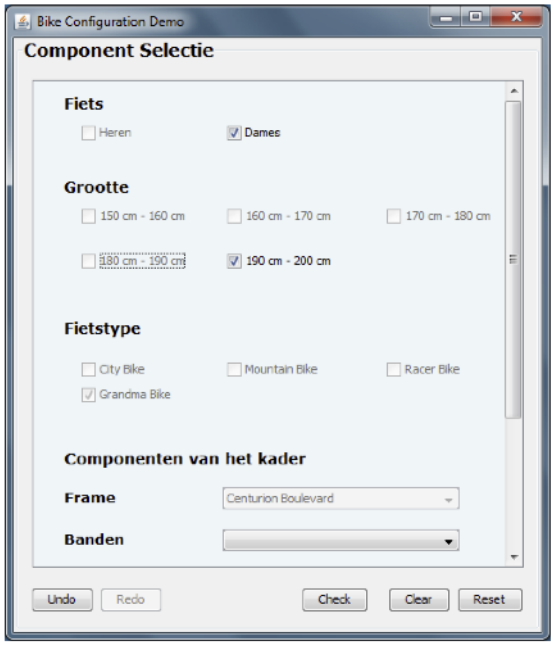

Figure 1: Screenshot of the API in action.

```
// Precisely one frame should be selected
#{x: SelFra(x)} = 1

// A carrier requires a mudguard
!x: BikeExt(x, Carrier) => BikeExt(x, Mudguard)

// You cannot have both a pump and a bottle
!x: ~(BikeExt(x, Pump) & BikeExt(x, Bottle))
```

Given information about the choices the user has made so far, the IDP system is then able to compute further consequences of this. In our demo, we use an approximate algorithm for this, which is not guaranteed to derive as much as possible in each step, but always runs in polynomial time.

Once the knowledge base and GUI have been constructed independently, the programmer's work is almost done, since the API allows him to link the two components with minimal effort. This is simply a matter of adding appropriate annotations to the UI elements, e.g.:

```
@IDPPredicate("SelFra")
private JComboBox jComboBox1;
```

This statement informs the API that the component `jComboBox1` of the GUI corresponds to the predicate `SelFra` of the knowledge base. Once it knows these correspondences between GUI components and the logical vocabulary of the knowledge base, it can automatically take care of all communication between the GUI and the knowledge base by itself.

References

[1] The IDP system. `http://dtai.cs.kuleuven.be/krr/software/idp`.

[2] Nick Calus. Een java api voor een kennisbanksysteem. Master's thesis, Lessius Mechelen, Campus De Nayer, 2010-2011.

[3] Hanne Vlaeminck, Joost Vennekens, and Marc Denecker. A logical framework for configuration software. In *Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming PPDP '09*. ACM, September 2009.